LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
Il6r
no. 279-286
cop. 2

[redacted]

CENTRAL CIRCULATION BOOKSTACKS

The person charging this material is responsible for its renewal or its return to the library from which it was borrowed on or before the **Latest Date** stamped below. **You may be charged a minimum fee of $75.00 for each lost book.**

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

TO RENEW CALL TELEPHONE CENTER, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

9/17/96

SEP 0 6 1996

When renewing by phone, write new due date below previous due date. L162

Digitized by the Internet Archive
in 2013

http://archive.org/details/illiacivquarterl286univ

I26r
no.286

Math

Report No. 286

ILLIAC IV

QUARTERLY PROGRESS REPORT

April, May and June, 1968

Contract No.
US AF 30(602)4144

THE LIBRARY OF THE
[illegible] 1968
UNIVERSITY OF ILLINOIS

ILLIAC IV Doc. No. 203

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

ILLIAC IV

QUARTERLY PROGRESS REPORT

April, May and June 1968

Contract No.
US AF 30(602)4144

Department of Computer Science
University of Illinois
Urbana, Illinois
61801

July 1, 1968

This work was supported in part by the Department of Computer Science, University of Illinois, Urbana, Illinois, and in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center, under Contract No. US AF 30(602)4144.

TABLE OF CONTENTS

1. REPORT SUMMARY

The power distribution problem has been solved by placing shunt regulators in the PU cabinets. This provides the regulation after the connector rather than before, and thus should eliminate regulation problems.

Orders were placed for additional magnetic tape units and disk modules for the B5500. The additional equipment will provide the configuration necessary to run ILLIAC IV design automation programs on the University's B5500. The B5500 has been running very well during the last several months. This is attributed to efforts by the University operating staff and increased support by Burroughs.

A data link connecting the University's B5500 with Paoli's 5500 has been ordered and should be installed in August.

A contract has been let with Mr. Joseph "Bud" Wirsching, who is with Information Systems Design, Incorporated, to study the application of ILLIAC IV to weapons effect calculations.

Additional educational material is being developed and classes were held for new graduate students joining the program during the summer.

Representatives of the ILLIAC IV project attended the ARPA Network meeting at the University of Illinois in July to discuss ILLIAC IV's role in the ARPA Network.

Preliminary installation plans for ILLIAC IV have been approved by the University and Burroughs is in the process of generating final drawings.

The breadboard PE was delivered to Paoli on June 11. Burroughs plans to debug the PE breadboard and the PE exerciser, simultaneously. Cooperation between the University diagnostic group and Burroughs has been established and test programs will be generated by the University group.

2. HARDWARE

2.1 Diagnostics

2.1.1 Diagnostic Generator System

The PE Diagnostic Generator System has been under programming activity during this period. The first version of the System consists of the following programs and will generate diagnostic programs for the breadboard PE which is being debugged at Paoli. All the programs are written in the Extended ALGOL for the B5500.

1) Control Signal List Editor
2) Path Generating Method
3) Test Ordering Program
4) Path Test Generator
5) Combinational Test Generator
6) PEXTAP Input Generator for Path Tests and Combinational Tests
7) Wiring List Editor
8) Test Simulator Generator
9) Test Simulating Program
10) PEXTAP Input Generator for Functional Tests
11) PEX Test Assembling Program
12) Documentation Generator for Net List and Placement

Connections among the programs of the Generator System are shown in Figure 1.

Burroughs has been writing the last three programs including the PEX Test Assembling Program which was extended to have more documentation facilities.

A few programs for test simulation were added to the System in June to make the most of a slippage in building the breadboard PE. Another program, the Control Signal List Generator, which was to be included in the System, has not been written for the first version.

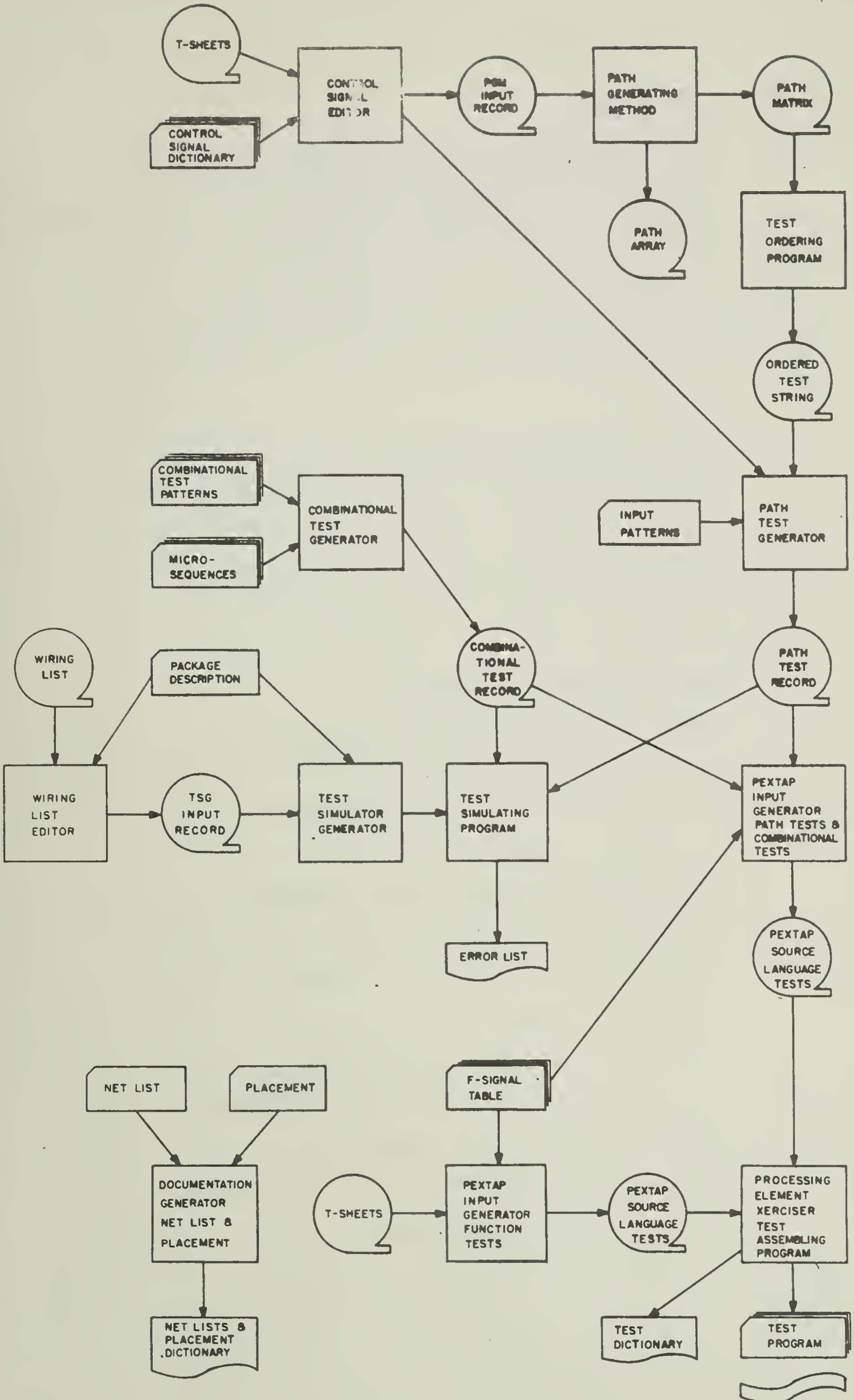

T-SHEETS
CONTROL SIGNAL DICTIONARY
CONTROL SIGNAL EDITOR
PGM INPUT RECORD
PATH GENERATING METHOD
PATH MATRIX
PATH ARRAY
TEST ORDERING PROGRAM
ORDERED TEST STRING
COMBINATIONAL TEST PATTERNS
MICRO-SEQUENCES
COMBINATIONAL TEST GENERATOR
INPUT PATTERNS
PATH TEST GENERATOR
WIRING LIST
PACKAGE DESCRIPTION
COMBINATIONAL TEST RECORD
PATH TEST RECORD
WIRING LIST EDITOR
TSG INPUT RECORD
TEST SIMULATOR GENERATOR
TEST SIMULATING PROGRAM
PEXTAP INPUT GENERATOR PATH TESTS & COMBINATIONAL TESTS
ERROR LIST
PEXTAP SOURCE LANGUAGE TESTS
NET LIST
PLACEMENT
F-SIGNAL TABLE
DOCUMENTATION GENERATOR NET LIST & PLACEMENT
T-SHEETS
PEXTAP INPUT GENERATOR FUNCTION TESTS
PEXTAP SOURCE LANGUAGE TESTS
PROCESSING ELEMENT XERCISER TEST ASSEMBLING PROGRAM
NET LISTS & PLACEMENT DICTIONARY
TEST DICTIONARY
TEST PROGRAM

FIGURE 1. DIAGNOSTIC GENERATOR SYSTEM FOR BREADBOARD PE

2.1.2 Program Status

2.1.2.1 CSLE (Control Signal List Editor)

The CSLE was developed to check control signals in the T-sheets prepared by Burroughs logical designers, to generate some listings for the benefits of the designer and the diagnostician, and to generate an arc description to be input to the PGM program. This program requires about an hour for execution on the B5500.

2.1.2.2 PGM (Path Generating Method)

The last errors were removed from and a few modifications have been done to the PGM program in this period. The program requires about 25 minutes for 800 arcs. One hundred ninety-four test paths have been produced for the first version tests of the breadboard PE.

2.1.2.3 TOP (Test Ordering Program)

Three subprograms of the TOP were written and tested with PATH MATRIX. The programs that have been written are PATH MATRIX WITH SUFFIX DELETED NODES, REDUCED PATH MATRIX, and TABLE OF MASKED NODES.

PATH MATRIX WITH SUFFIX DELETED NODES: The original PATH MATRIX has 600 nodes. After adding suffix-deleted nodes, the number of nodes increases to 694. The number of paths remains the same.

REDUCES PATH MATRIX: This program was written to delete equivalent nodes and redundant nodes. Redundant nodes are those nodes which have zero column vectors. After the deletion of equivalent and redundant nodes, the number of nodes of the path matrix decreased to 416. The reduced path matrix is the path matrix for path ordering.

TABLE OF MASKED NODES: The path matrix is packed into an alpha-array. In order to save time in finding the masked nodes, word-with-word comparisons are used first, and then bit-with-bit comparisons are used. The table of masked nodes is contained in a 2-dimensional, 5 x 1000, array called MASK and two other 1-dimensional arrays called NØSMALL (No. of smaller nodes, i.e., masked nodes) and PNTER (pointer).

The i-th element of array PNTER tells the starting nodes in array MASK, masked by the i-th nodes of reduced path matrix. The i-th element of array NØSMALL tells the number of masked nodes of i-th node of reduced path matrix.

The last subprograms, THE SET OF EQUIVALENT MULTIPLE FAULTS, (SEMF), and THE ORDERING OF PATH TEST, are being written and tested.

2.1.2.4 PTG (Path Test Generator)

The PTG program is divided functionally into two subprograms--segmenting and editing and expected response calculation.

The outputs from the PGM program and the TØP are analyzed and reformed into actual test sequences. Test strings of the output data of PGM may contain loops. When a test path passes through the same node two or more times in a string, this is segmented at the appropriate nodes. Also, some editing of the test strings is required for the output data of PGM program.

The expected response calculator for the path tests has been written. The writing of the expected response calculator was based on the control signal dictionary containing specifications of data paths.

The PTG has been written and has been working during this period. The expected response calculator may be modified according to design changes.

2.1.2.5 CTG (Combinational Test Generator)

In this quarter, the Combinational Test Generating program (CTG) became operational. CTG is used to expand abbreviated descriptions of test patterns and test sequences into a full set of test patterns and test sequences in the PEX Assembly Language. Descriptions of microsequences are expanded into a list of F-signals for translation into an octal code by another program (PIG). CTG also has provisions for inserting comments and PEX instructions into the Assembly Language code.

For each pattern generated, CTG calculates PE expected response and outputs this data along with the pattern for comparison with the actual (possibly erroneous) response of the PE. Procedures for calculating expected response of CPA, LOD, CSA, and ADA are presently operating. Expected response for CPA, LOD, and ADA is determined by a functional simulation of these combinational blocks. In the case of CSA, this calculation is accomplished by a detailed logic simulation for one clock cycle of the multiply operation. The expected response procedure for BSW is being added.

CTG has been used to generate a set of microsequences, patterns, test sequences, and PEX instructions for CPA testing. Additional runs for the remaining combinational tests will be made at the beginning of the next quarter.

2.1.2.6 PIGPC (PEXTAP Input Generator for Path Tests and Combinational Tests)

The main function of the program is to translate the control signal names in Path/Combinational Test Record into a fixed format for the PEXTAP. This program was written during this period.

2.1.2.7 WLE (Wiring List Editor)

The WLE works on the PE wiring list to produce a set of inter-package arcs to be input to the Test Simulator Generator and to produce some files and a listing of detected errors. The program was written and was used to generate 1797 arcs among 431 packages (or circuit cards).

2.1.2.8 TSG (Test Simulator Generator)

The main function of the TSG is to assign levels to the packages, taking into account the chains or sets of connected loops of packages, and to generate the simulator body referring to the ordering. The TSG has been written and is being tested.

2.1.2.9 TSP (Test Simulating Program)

A primary simulating program was generated by the TSG and was run with a sample test program. Some debugging must be done before it becomes operational on both Path and Combinational Test Records.

2.2 Design Automation

The Design Automation Group has started a program to determine the signal delay between the clocked elements on the CU boards. Because of the speed of the ECL logic elements, the total of the gate delay and a nominal amount of wiring delay is not an adequate estimate of the signal delay between clocked elements. After the circuit boards have been laid out and a router solution obtained, the total delay will be determined by the delay check program.

The initial specifications of the program and the basic technique are complete, and the programming has begun. The program will operate as part of the Burroughs Design Automation System when completed.

3. SOFTWARE

3.1 Translator Writing System

3.1.1 TWS Syntax

The Floyd production parser and the syntax preprocessor were thoroughly tested and debugged using the syntax for Tranquil, Glypnir, and many other minor languages and using source programs in these languages. Although successful parsing of source programs was achieved, these tests indicated many inefficiencies in the system. Plans for greatly improving the efficiency of both the parser and the syntax preprocessor were made, and they are currently being rewritten to implement these more efficient techniques. Also, the plans call for more sophisticated parser instruction table generation in the syntax preprocessor, which are being implemented concomitantly with the new parser. By using the test programs, new plans for improved error recovery and improved user interface (output messages) were conceived and are also being implemented in the new parser.

3.1.2 TWS Semantics

During this quarter, the work in the semantic area of the TWS can be divided in three areas:

a) The preparation of the new ISL translator, using the TWS system itself (as described in the last QPR) is now well advanced. This new ISL translator (called in this report the TWS-ISL translator) will eventually be the standard translator since, due to the use of the TWS, it will be easily modifiable by any user familiar with the system.

b) It is expected that the TWS-ISL translator will, at first, be very slow (about 30 cards per minute on the average) compared with the present (brute-force) translator (about 200 cards per minute, on the average). As the parser is improved, the speed of the TWS-ISL translator will increase until it becomes fast enough to replace the old translator. However, until this is achieved, the brute-force-ISL translator will be our main ISL translator, and considerable time was dedicated to its improvement. Control-cards and modifications were added to make it "n-parse compatible" as mentioned in the previous QPR.

c) Since the speed of 200 cards per minute obtained with the old ISL translator is still not very satisfactory, a major effort is being conducted to modify it in order to obtain a considerable improvement in the speed.

The work in the three areas described above is being conducted in parallel. Coding is almost completed, and debugging has begun in all areas.

3.2 Tranquil

During this quarter, the writing of the Pass 1 semantics was essentially completed and debugged. Work was begun in writing the Pass 2 semantics, and the skeleton structure of Pass 2 and a number of utility routines for outputting assembly language were written. Finally, a more sophisticated program to integrate inherently sequential statements into a form suitable for parallel execution was written.

3.3 Glypnir

The work on Glypnir is continuing on schedule. It is expected that the Version I compiler will be running by September.

Conversion of syntax and semantics from the Floyd Production to the recursive descent compiler-compiler system was completed with a minimum of difficulty. Work is currently progressing in parallel using both systems. A decision as to which system will be used for Version I will be made shortly.

The semantic code for operand fetches, arithmetic expressions, and assignment statements has been completed. Work is continuing on conditional statements and Boolean expressions.

Syntactic debugging of Glypnir programs should be possible by July 15, and a programming manual should be available during August.

3.4 Debugging System (SQUASH) for B5500 ALGOL

The SQUASH system is an extension of B5500 ALGOL. The SQUASH extensions control ALGOL programs in several ways which should be of help in debugging. SQUASH commands (or statements) include PRINT, TRACE, and other special commands or ALGOL statements. A SQUASH command, which is a separate ALGOL statement, defines a section of the user's ALGOL program as its scope and defines actions to take place at various points in the program within that scope. When normal execution of the program reaches the beginning of the scope of a SQUASH command, the following take place: 1) the section is executed once with the various defined actions also executed; 2) the program variables are reset; and 3) the program section (scope) is executed normally.

The syntax specification for the TWS syntax preprocessor is almost complete. Semantic actions have been formulated but have not been coded.

Codes for scanning the initial input file of user program and SQUASH statements have been begun. These codes are used to build tables of variable names, types, arrays' dimensions, block structure, and other information about the user's program. These tables are then used by the semantic actions corresponding to the parsing of the SQUASH syntax to produce ALGOL code. The resultant code is then merged with the original ALGOL program to create the final product.

Some new semantic ideas have been considered, especially with regard to the resetting of program variables after the execution of SQUASH actions during program execution.

3.5 CAT

The CAT compiler can be broken into several vaguely distinct subsystems which provide the necessary processing to reduce a Tranquil II code, written for an arbitrary parallel machine, to a set of Tranquil I programs with control and I-O connections which will run on ILLIAC IV. The first of these subsystems is the geometric processors. A general Tranquil II code will contain a <geometry block> which defines the arrays with which the remainder of the code deals. Syntax and semantics have been developed for such blocks. (For the specifics of the syntax and semantics, see Appendix A.)

The geometry processor will analyze the information in this geometry block and will produce an index set or mode pattern for each geometric identifier which appears later in the body of the code. The implication is that the region described in an almost mathematical way in the geometry block will be converted to a form that the rest of the CAT compiler can understand.

At this point, the mathematical algorithm used in the code will be examined by a simple linear programming algorithm which will reorder the computation in such a way that the number of I-O transactions is minimized. At the same time, the storage allocation and I-O requirements will be determined, and the data will be blocked properly.

Finally, an I-O transaction and control table will be constructed, and the various kernels will be compiled by the Tranquil I compiler using the index set constructed by the code and data slicer.

At execution time, these kernels plus a CAT control program, which monitors and maintains the I-O and program flow tables constructed by the compiler, will be loaded into the ILLIAC IV-B6500 system and the code will be run.

3.6 SYSTEM K

3.6.1 Introduction

The activities of the SYSTEM K group were oriented around the following:

1) ILLIAC IV simulator and basic assembler,

2) Operating system,

3) B5500 software maintenance, and

4) SDU and CUBE.

3.6.2 ILLIAC IV Simulator and Assembler

The simulator is in the process of being revised according to the new operation code specifications and will be re-released in two versions. The first version (due 1 August) will be an efficient, 1-quadrant simulator with disk I-O intrinsics. The main users of this simulator will be applications' programmers who want to test their programs in the least amount of time. The second version will be a four-quadrant simulator that will simulate multi-quadrant operation of the ILLIAC IV.

The basic assembler has been increased in efficiency and will now assemble over 700 cards per minute. The second pass of this assembler will be the last pass of the macro assembler.

3.6.3 ILLIAC IV Operating System

During the last quarter, much study has been devoted to the B6500 side of the B6500-ILLIAC IV interface. A liaison between the SYSTEM K group and Burroughs, Pasadena, (which is developing the B6500) has been established. The available information indicates that all the basic tools that are needed to write the ILLIAC IV operating system will be part of the Burroughs standard software.

Work has begun on the ILLIAC IV resident operation system (OSK4). Disk I-O commands will be available and operative with the 1-quadrant simulator due 1 August.

Progress is being made on the services sub system (SSS). A multi-teletype message switcher was written and demonstrated. This program communicates with several teletypes simultaneously and will route a message to a designated user--if the user is at a teletype, he will receive the message immediately; if not, the message is stored in a disk file for him.

3.6.4 B5500 Maintenance

The B5500 is now running smoothly and much more reliably. A program which isolated a character-changing disk bug was written, and the bug was fixed. The MCP itself is now fairly reliable and has survived times of incredible overload without halting.

The log routines are successfully generating reports for internal accounting.

3.6.5 SDU and CUBE

A program which enables a programmer to use the SDU (the experimental CRT terminal lent to the project by Burroughs) for editing disk files was developed. This program (REMOTE/SDU) is a modification of REMOTE/CARD written by Ron Brody of Burroughs, Paoli.

The CUBE (Cooperating Users of Burroughs Equipment) library was procured. Among the programs in this library was TEACHER. TEACHER was designed for computer assisted instruction using the teletype and accepts input in the IBM course writer language. A short course has been written to explain what the ILLIAC IV is. It is planned to write courses on the use of the B5500 and the ILLIAC IV assembler and simulator. These courses will become part of the orientation and training program for new members of the ILLIAC IV staff.

4. APPLICATIONS

4.1 Mathematical Applications

4.1.1 Differential Equations

4.1.1.1 First Order Partial Differential Equations

A difference scheme was developed to solve a system of two coupled nonlinear first order partial differential equations. The Von Mises transformation was employed to reduce the two equations to one quasilinear second order partial differential equation. The present problem arises in studying the flow past a flat plate with sharp leading edge. At the wall, the Maxwell slip boundary condition is used.

For the specifics of this difference scheme, ILLIAC IV Document Number 189 can be consulted [1]. The comparison of the results of the difference scheme with the available theoretical and experimental data is also included in the document.

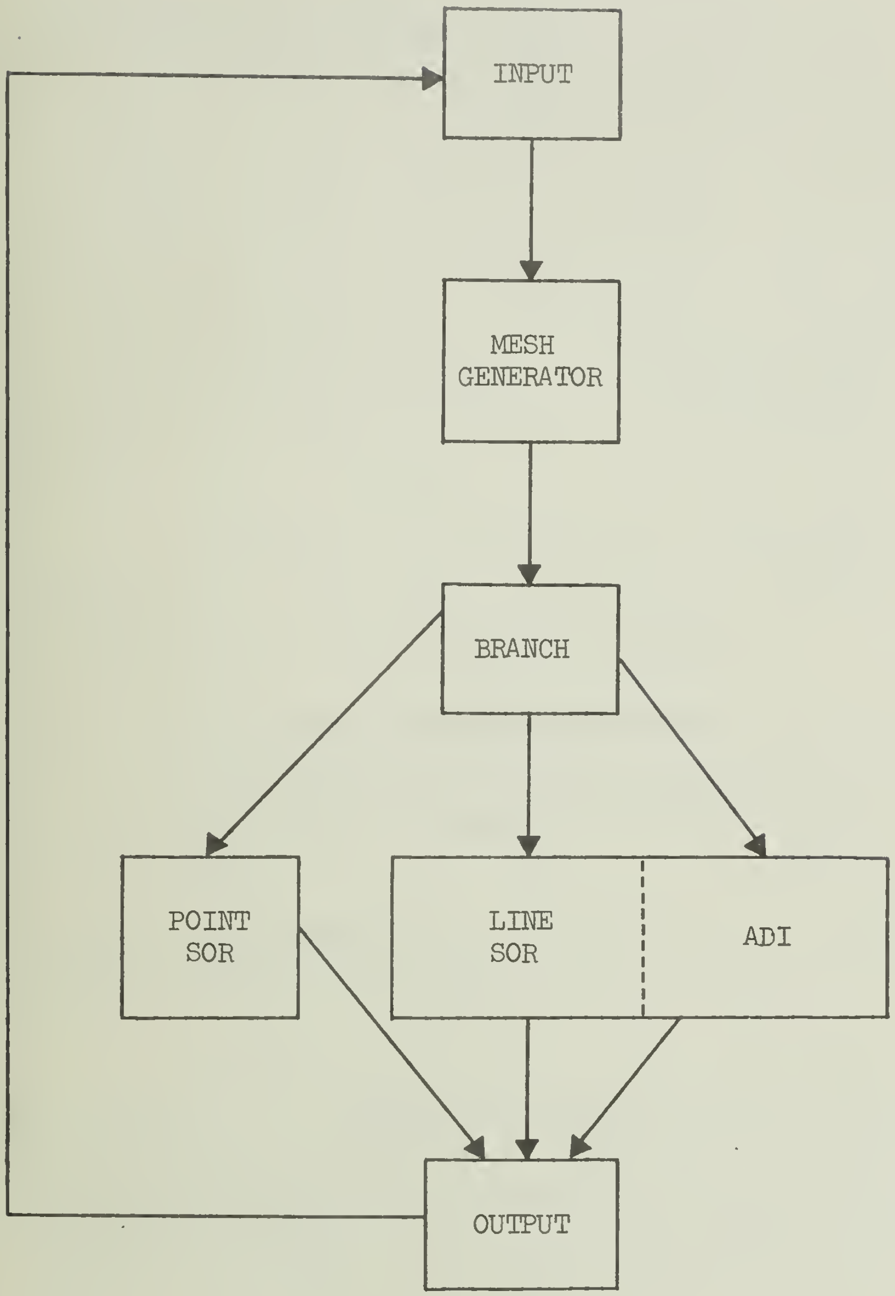
4.1.1.2 Elliptic Partial Differential Equations

The two-dimensional heat equation which was coded last summer is completely updated and free of syntax errors. The three-dimensional heat equation which was written last fall is in the process of being updated and should be completed shortly.

Time was also spent on developing a large block of code that will be used to solve general two-dimensional elliptic partial differential equations. The following flow chart will give a general idea of what the block of code looks like.

The following outline will describe more fully the functions of each of the boxes in the flow chart.

I. Input
   A. Boundary conditions
      1. Tabulated values
      2. Functions



Code Flow Chart for General Two-Dimensional
Elliptic Partial Differential Equations

        3. Constants
        4. Dirichlet and/or Neumann boundary conditions
    B. Description of the Region and mesh size
    C. Control information
        1. Iteration parameters
        2. Convergence criteria
        3. Logical flags
        4. Print out control
    D. Initial guess

II. Mesh Generator
    A. Setting up the mesh
    B. Setting up tables of indices
    C. Loading boundary values

III. Branch

    Determination of the method to use from input or the mesh generator

IV. Point Successive Overrelaxation (PSOR)

V. Line Successive Overrelaxation (LSOR)

VI. Alternating Direction Implicit method (ADI)

VII. Output

Because certain procedures for LSOR and ADI are very similar and can be shared by the two methods, the LSOR and ADI methods are contained in the same box in the flow chart.

4.1.1.3 Partial Differential Equations and CAT

The problem under investigation is that of slicing data for PDE problems to be handled by CAT. The necessity of slicing meshes arises for non-core-contained problems, and its aim is to minimize latency in I-O to the disk. For the two-dimensional case, one very straightforward scheme is slicing the mesh into large square blocks. The latency for this scheme is slightly more than 25%, depending on mesh size parameters. It is noted that for cases in which updating calculations are longer than I-O time, this 25% latency may not be a deficiency.

Another scheme exists which is actually a family of schemes. One of this family is chosen, depending on problem size. Meshes are sliced into small segments as opposed to the large blocks of the scheme mentioned above. For some problems approaching disk capacity, this scheme cannot be used; however, for the much wider range of medium-large to small problems this scheme promises to be quite efficient. Latency has been limited to 25%, but a large number of problems can be handled with zero latency. More than half of all possible rectangular problems can be handled with less than 10% latency. There are possibilities to modify this scheme to suit problem calculation times exactly.

Problems with three-dimensional meshes have not yet been studied, but the "small segment" scheme can be used for 3-D problems with a small third dimension (e.g., 25 points).

4.1.1.4 Simultaneous Differential Equations

Work has been proceeding on solving the set of simultaneous differential equations of the form:

$$X_K' = \sum f_{ij}^{(K)} X_i X_j + \sum g_{ij}^{(K)} X_i(t - \tau)\, X_j(t - \tau) + \sum c_i^{(K)} X_i$$

A program has been coded in ILLIAC IV Assembly language. This program works on the present assembler and simulator on the B5500; although, of course, it is significantly slower than it will be on ILLIAC IV itself. A timing simulation has been done which indicates that this program could execute 1500 iterations per second on ILLIAC IV. This does not include I-O time.

An ALGOL program of the problem was also coded, and the results of the Assembly Language program and the ALGOL program were compared. After some debugging and after the determination of the truncation error, identical results were obtained. A Tranquil coding is planned along with further analysis of the problem.

4.1.1.5 Codes for Differential Equations

4.1.1.5.1 Code for Hydrodynamic Equations

During the last quarter, a Tranquil code solving the Eulerian hydrodynamic equations in two-dimensional cartesian coordinates using "checkerboard" storage was completed. Since 5 hydro quantities are stored in core for each cell in the mesh, the "checkerboard" storage scheme will allow a mesh consisting of up to approximately 100,000 cells to be stored in core.

A Tranquil code which traces the path of "mass-less" particles through an Eulerian mesh is nearing completion. The Eulerian hydro code and the "mass-less" particle trace code will be described in a forthcoming document.

4.1.1.5.2 Code for Hockney's Method

The assembly language code for Hockney's [2] method for solving Poisson's equation is being run on the simulator. A timing estimate for the code will be obtained from the Sankin simulator. The Tranquil version of Hockney's method has been syntactically debugged and will be run on the Tranquil compiler as soon as the compiler is ready. Results from the two codes will serve as a comparison between machine language and Tranquil.

4.1.1.5.3 Code for Boltzmann's Equation

Implementing the evaluation of the Boltzmann [3] collision integral to ILLIAC IV was begun during the last quarter. An efficient storage scheme has been devised to core contain the problem, and a Tranquil code for the problem is now being written.

4.1.1.6 Hydrodynamic Calculations*

In solving the Eulerian hydrodynamics equations numerically, a grid of rectangular cells is superimposed over the region of fluid

*This area has been similarly printed as an ILLIAC IV Document. See [4].

under consideration. The resulting difference equations require access to the four nearest neighbors for each cell in the grid. Several boundary adjustments, which should also be considered in the allocation of storage, are necessary in the scheme. The ideal is that all the available processing elements (PE's) would simultaneously adjust the boundary variables. Later, in the particle description of the fluid, it is important that the particle motion across cell boundaries does not overload any single processing element memory. The assignment of cells to PE memories should also take into account the distribution of particles throughout the grid to insure maximum processing efficiency.

Therefore, the four conditions to be considered in the assignment of storage to the hydrodynamic variables are:

1) access to four nearest neighboring cells,

2) efficiency boundary adjustment,

3) no memory overload in particle motion, and

4) efficient distribution of cells among PE memories.

Essentially, the PE's are skewed in groups of nine PE's, and the 9 x 9 groups or blocks are themselves column-skewed with internal skewing by rows within a block. For convenience, let this method of storage be called the internal block skewed form.

A brief comparison of internal block skewing with the checkerboard form of storage is useful. Four blocks of a grid in checkerboard storage are illustrated below. If I and J represent the coordinates in an orthogonal two-dimensional system, then the processing element which contains the cell (I,J) in checkerboard form is given by PE = [I + (J-1) x 16] mod 256.

1	2	3	4	5	6	7	8	9	17	18	10	11	12	13	14	15	16	24	111	119	120	121	122	123	124	125	126	118
9	1	2	3	4	5	6	7	8	16	17	18	10	11	12	13	14	15	23	110	118	119	120	121	122	123	124	125	126
8	9	1	2	3	4	5	6	7	15	16	17	18	10	11	12	13	14	22	109	117	118	119	120	121	122	123	124	125
⋮									⋮									•••••		⋮								
3	4	5	6	7	8	9	1	2	10	11	12	13	14	15	16	17	18	26	113	121	122	123	124	125	126	118	119	120
2	3	4	5	6	7	8	9	10	18	10	11	12	13	14	15	16	17	25	112	120	121	122	123	124	125	126	118	119
10	11	12	13	14	15	16	17	18	26	18	19	20	21	22	23	24	25	33	120	128	129	130	131	132	133	134	126	127
18	10	11	12	13	14	15	16	17	25	26	27	19	20	21	22	23	24	32	119	127	128	129	130	131	132	133	134	135
17	18	10	11	12	13	14	15	16	24	25	26	27	19	20	21	22	23	31	118	126	127	128	129	130	131	132	133	134
⋮									⋮									•••••		⋮								
12	13	14	15	16	17	18	10	11	19	20	21	22	23	24	25	26	27	35	122	130	131	132	133	134	135	127	128	129
11	12	13	14	15	16	17	18	19	27	19	20	21	22	23	24	25	26	34	121	129	130	131	132	133	134	135	127	128
⋮									⋮																			
⋮									⋮																			
235	236	237	238	239	240	241	242	243	251	243	244	245	246	247	248	249	250	2	89	97	98	99	100	101	102	103	104	96
243	235	236	237	238	239	240	241	242	250	251	252	244	245	246	247	248	249	1	88	96	97	98	99	100	101	102	103	104
242	243	235	236	237	238	239	240	241	249	250	251	252	244	245	246	247	248	56	87	95	96	97	98	99	100	101	102	103
⋮									⋮									•••••		⋮								
237	238	239	240	241	242	243	235	236	244	245	246	247	248	249	250	251	252	4	91	99	100	101	102	103	104	96	97	98
236	237	238	239	240	241	242	243	244	252	244	245	246	247	248	249	250	251	3	90	98	99	100	101	102	103	104	96	97
244	245	246	247	248	249	250	251	252	4	252	253	254	255	256	1	2	3	11	98	106	107	108	109	110	111	112	104	105
252	244	245	246	247	248	249	250	251	3	4	5	253	254	255	256	1	2	10	97	105	106	107	108	109	110	111	112	113
251	252	244	245	246	247	248	249	250	2	3	4	5	253	254	255	256	1	9	96	104	105	106	107	108	109	110	111	112
⋮									⋮									•••••		⋮								
246	247	248	249	250	251	252	244	245	253	254	255	256	1	2	3	4	5	13	100	108	109	110	111	112	113	105	106	107
245	246	247	248	249	250	251	152	253	5	253	254	255	256	1	2	3	4	12	99	107	108	109	110	111	112	113	105	106

To illustrate the first alternate storage sheme, map the PE's onto a grid of size 252x 126 in a fashion shown above. The numbers indicate the PE memory in which the corresponding cell of the grid is located.

1 2 3 ... 15 16	17 18 19 ... 31 32
17 18 19 ... 31 32	33 34 35 ... 47 48
.	.
.	.
.	.
241 242 243 ... 255 256	1 2 3 ... 15 16
1 2 3 ... 15 16	17 18 19 ... 31 32
17 18 19 ... 31 32	33 34 35 ... 47 48
.	.
.	.
.	.
241 242 243 ... 255 256	1 2 3 ... 15 16

1) By using checkerboard storage, the nearest horizontal neighbors of a given cell can be accessed by a route of one, but the nearest vertical neighbors of a given cell are reached by a route of two. Internal block skewing handles nearest-neighbor accessing in one route for both directions.

2) The entire horizontal boundary can be adjusted simultaneously in checkerboard storage. However, vertical boundaries must be considered for 16 cells at a time. By internal block skewing, either boundary can be adjusted simultaneously as a whole.

3) No memory overload is anticipated in internal block skewing or checkerboard storage.

4) It is guessed that internal block skewing is better than ordinary skewing for decreasing the idle-time of a processing element during a hydrodynamics problem run. In this respect, however, it is felt that internal

block skewing would be less efficient than checkerboard storage; since in checkerboard storage, each 16 x 16 block contains every PE, each PE is given the chance to remain active.

Modifying the checkerboard scheme yields another form of storage. Instead of retaining the 16 x 16 blocks used in checkerboard storage, 32 x 8 blocks are substituted. Four such blocks are shown below. Here, the PE which contains cell (I,J) is PE = [I+(J-1) x 8] mod 256.

1	2	3 ...	7	8	9	10	11 ...	15	16
9	10	11 ...	15	16	17	18	19 ...	23	24
⋮					⋮				
249	250	251 ...	255	256	1	2	3 ...	7	8
1	2	3 ...	7	8	9	10	11 ...	15	16
9	10	11 ...	15	16	17	18	19 ...	23	24
⋮					⋮				
249	250	251 ...	255	256	1	2	3 ...	7	8

Advantages of this form over ordinary 16 x 16 block checkerboarding are that nearest neighbors for a given cell are one route away and the vertical boundary can be adjusted for 32 cells at a time. Comparing modified checkerboarding with ordinary checkerboard storage, processing efficiency is likely to be reduced for the modified checkerboard form, especially when the particle density is sparse.

4.1.1.7 Codes

During this quarter, there have been three Tranquil codes written for rectangular regions using Jacobi, Gauss Seidel, and SOR iterative methods. El-Assar's slipflow problem has also been programmed.

It is planned that by the end of the summer there will be two more codes completed. An ASK code is being written which uses the ADI iterative method on a square region. This code will be compared to an existing Tranquil code for the same method and region. Another code that may be completed by the end of the summer is a Tranquil code which will use SOR for semi-general regions. This code will be included in a larger code with other iterative methods.

4.1.2 Matrix Problems

4.1.2.1 Sparse Matrix Multiplication

An algorithm for the multiplication of two sparse matrices was developed during this quarter. To form the inner product of a row-vector a, stored in the CU, and a column vector b, stored in a PE, the algorithm forms a queue in the PE consisting of the non-zero elements of a that are to be used as operands and associated pointers to the non-zero elements of b which are to be used as operands. The algorithm then proceeds to step through the queue performing only those multiplications which result in non-zero products.

Since, in general, the queue will be a different length in different PE's, certain PE's will be inactive until the longest queue is processed. However, they are inactive only when the <u>total</u> number of multiplications is less than the maximum <u>total</u> number of multiplications. In other words, each PE is always performing its next multiplication regardless of the progress of the other PE's.

A storage scheme that allows storage of only the non-zero elements was also developed. The form of the original matrix is retained in mode words (defined in the paper referenced at the end of this section). The mode words are constructed so as to simplify the determination of which multiplications are to be performed.

The details of the algorithm and storage scheme are in ILLIAC IV Document Number 191, [5]. Also included in the document are the modifications of the algorithm for handling dense times sparse and sparse times dense matrix multiplication. A flow chart and a program in ILLIAC IV assembly language can also be found in the document.

4.1.2.2 Sparse Matrix Inversion on ILLIAC IV

A document discussing the inverse of a large (of order 500 to 1000) sparse matrix will be printed. It will describe an algorithm for the inversion of irregularly sparse matrices which are stored packed and with tags for the non-zero elements.

The processing of giant sparse matrices is difficult because only non-zero elements are stored in memory. The difficulty may be overcome by using the Expand Row Routine and the Pack Row Routine. The algorithms for these routines and the overall flow-chart will be included in the document as examples. Furthermore, in order to reduce operation time, the scheme in which the operations are eliminated for the rows having zeros in the pivot column is used.

An algorithm without pivot search written in PL/1 is as follows:

```
PIVOT:      DO K = 1 TO M;
            DO = A(K,K); A(K,K) = 1;
PIVOTROW:   A(K,*) = A(K,*)/D;
NONPIVOT:   DO I = 1 TO K - 1, K + 1 TO N;
            D = A(I,K); A(I,K) = 0;
            A(I,*) = A(I,*) - D*A(K,*);
   END PIVOT;
```

A program is being coded in the ILLIAC IV's assembler language. The program will be tested on the assembler and the B5500 simulator, and a timing simulation will also be done.

4.1.3 Math Subroutines

4.1.3.1 Root Finding

The adaptation of Lehmer's method for use on the ILLIAC IV was accomplished this quarter. The complex roots of a polynomial with coefficients are found by Lehmer's method, after the simple real roots have been determined by the multisectioning method. Lehmer's method determines if a given polynomial f(x) has any roots in a circle with center c and radius p. An explanation of the method follows.

First, Gerschgorin circles are used to find a bounded area in which all of the roots can be found. If $f(x) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_0$, then all of the roots of f(x) are found in a circle with $c = 0$ and $p = \max|a_i/a_n|i = 0, 1, \ldots, n-1$. The square circumscribed around the circle can therefore be subdivided into 256 squares. Next, Lehmer's test for determining the presence of a root can be applied to the circles circumscribing each of the 256 squares. In each of the circles in which a root is found, the process is repeated. Since the squares are adjacent, their circumscribing circles intersect (4/7 of the total area is covered by intersection areas). The intersecting could cause the method to look for the same root from two different circles; however, this problem can be removed by deflating the polynomial whenever a root is found.

4.1.3.2 Special Functions Library

This quarter's work has continued on the ILLIAC IV Special Functions subroutine library. The assembly language codes which were written in ASK Version I for the 64-bit Sine, 64-bit Cosine, 64-bit Exponential, 64-bit Square Root, and 32-bit Square Root subroutines have all been successfully simulated by the current version of the simulator.

Timing studies on the Sankin Timing Simulator have been completed for the 64-bit subroutines. The results are as follows:

64-bit Sine	Maximum time	=	11.7 microseconds
64-bit Cosine	" "	=	10.9 microseconds
64-bit Exponential	" "	=	11.2 microseconds
64-bit Square Root	" "	=	16.0 microseconds

A document explaining the above codes has been written and will be printed shortly. The document includes the algorithm, the flow chart, the assembly code, and a summary of the timing studies for each of the above subroutines.

4.1.4 Matrices

4.1.4.1 Eigenvalue Problems

The adaptation or modification of Jacobi's and Householder's methods and QR-algorithm was the continued area of concentration for this quarter. The detailed discussion of this subject and the flow charts and Tranquil language programs for the algorithms can be found in ILLIAC IV Document Number 182, [6]. Some specific comments on each of the three methods follow below.

Although Jacobi's method proved to be one of the most effective for solving algebraic eigenvalue problems on a parallel computer, it is limited to symmetric matrices with dominant principal diagonals. Once this method converges, evaluation of the eigenvalues and eigenvectors is rather straightforward.

By means of elementary Hermitian orthogonal matrices, Householder's method reduces the symmetric matrix to the tridiagonal form. When the matrix is in the tridiagonal form, the evaluation of the eigenvalues and eigenvectors is performed by almost parallel computational techniques.

The QR-Algorithm is used for finding the eigenvalues of unsymmetric matrices. To obtain the best results, the matrix is first reduced to an upper Hessenberg form by using Householder's method. Convergence is accelerated by performing a single origin shift; however, when the matrix has complex conjugate eigenvalues, a double origin shift is used to avoid complex conjugate shifts.

4.2 Linear Programming

During this quarter, the evaluation of algorithms for the first linear programming system has been completed.

It has been decided that the initial system will be designed to solve problems of up to about 4000 rows. A second package, capable of solving problems up to about 30,000 rows will be developed during the next year. The two systems will be known as LPS (small linear programming system) and LPL (large linear programming system) respectively.

LPS will use a modification of the Revised Simplex product-form algorithm which will provide for multiple pricing of up to 100 vectors. Current timing estimates indicate that a 4000-row, 10,000 column, five (5) percent dense LP matrix will be solved in 2-3 minutes using this algorithm.

In addition to the basic solution algorithm, facilities will be provided in the code to allow the user to use price ranging on solution vectors, parametric programming on objective function and right-hand side, parametric variation of columns and rows in the matrix, ranges on the constraints, and bounds on the variables of the problem. The capability for the specification of multiple right-hand sides and objective functions will also be provided. Since many LP users want to obtain integer or mixed-integer solutions to problems, an attempt will be made to provide this capability (which is not available in most LP codes) by the definition of logical bounds on the levels of solution variables.

Considerable flexibility will be provided for the sophisticated user of the LPS code. A control language consisting of macro-statements such as <PRIMAL>, <DUAL>, <INVERT>, and similar solution strategy macros which he considers to be optimal for his problem will be developed. The user-written control-language program will be compiled and will result in the selection of the specified set of subroutines from the "canned" package, which will generate a solution using the required strategy.

Development of the mathematical and data-processing procedures necessary for implementation of the above design requirements is currently underway. Results will be reported in the next Quarterly Progress Report.

4.3 Computer Graphics

The predominance of computation time on ILLIAC IV will be involved with the manipulation of a defined mesh associated with a physical model; such as, the weather or weapons effect simulation. A program to automatically produce graphical images from this type of data as a standard output procedure was begun this quarter.

The goal of the program is to communicate to the investigator the results of the computation performed upon the mesh in their most meaningful form. It is felt that the most direct and qualitative insight into the computed data would be to view it as a three-dimensional, opaque, continuous surface. This will be achieved in the following manner. If the mesh under consideration does not happen to be triangular, it will be transformed into a triangularly connected mesh. The value of each mesh point is then used as a scalar of the altitude. This procedure will approximate the continuous surface representation of the data by small triangular facets--all the apexes of these triangles are lying on the surface defined by the data. This model will then be passed on to a program to produce the final image. That program will have the following capability: (1) continuous gray-scale toning, (2) hidden-plain elimination, (3) perspective, (4) contouring, (5) labelling, (6) color, and (7) stereo.

The Warknoch algorithm for hidden-plain elimination and shading is being studied as a basis for that aspect of the graphics program. The algorithm exhibits a great potential for parallel execution and is being analyzed for implementation on ILLIAC IV. Philippe Loutrel's work in this area has been noted, but it does not display the same potential feature of parallelism.

4.4 <u>Fingerprint Pattern Studies</u>

Work has continued on algorithms to extract reliable ridge direction information from a digitized fingerprint image along with a measure of reliability or confidence level for each directional value obtained. Digitized images of sufficient resolution were obtained from the CHLOE flying spot scanner system at Argonne National Laboratory, through the efforts of Mr. B. Shelman of the Argonne staff. However, usable working data has been difficult to obtain because of problems in format conversion and noise apparently introduced in the scanning process. A sharp increase in the activities of the Graphics and Diagnostics groups has necessitated temporary curtailment of the studies in this area.

4.5 Radar Processing for Ballistic Missile Defense

The efforts for the last three months have been concentrated in the areas of Kalman Filtering for tracking, signal analysis for designation and discrimination, and evaluation program (NISIM).

The Kalman Filter technique for tracking of missiles has been programmed for ILLIAC IV in assembly language and has been run on the timing simulator. As predicted in the previous progress report, the Kalman Filter technique takes much more memory space and time than the least square as described in ILLIAC IV Document Number 173, "Application of ILLIAC IV to Urban Defense Radar Problem," [7]. The correlation of new returns with predicted values in each PE will be performed similarly to the technique described in the report with the exception that the correlation will be done on EAR instead of XYZ and with means of varying the limits on R to get the best match. The amount of data saved in each target consists of a 7 x 7 matrix and a 4-element vector; however, during both the updating or corrector stage and the integration stage, a large amount of working storage is needed. Approximately forty to fifty targets per PE can be stored in each PE; however, possible schemes of using backup storage for track tables are being investigated. Each PE can handle approximately one hundred targets from a running time standpoint. If the number of targets remains similar to those assumed in the report, it will be necessary to have a two-quadrant system if the Kalman Filter is used for tracking.

These programs are going to be run on the running simulator to actually see how ILLIAC IV would perform these codes. Some missile track data has been obtained from Lincoln Labs along with the results obtained by running the data on a standard computer using Kalman Filtering. This will be used to evaluate the correctness of the programs written for ILLIAC IV.

The Kalman Filter program is also being written in the Tranquil language to evaluate how efficient it will be to program the radar problem in the higher level language. When the compiler is completed, this program will also be run on the simulator to evaluate the overhead time wasted in programming in the higher level language as compared to the assembly language versions.

The efforts in signal analysis have been oriented towards finding information on techniques used in this area for designation and discrimination. Most techniques seem to utilize a Fourier analysis of the returns signals from the objects. The use of the Cooley-Tukey algorithm for making the Fourier analysis has been investigated for implementation on the ILLIAC IV. A report is being generated to describe this effort in detail and to relate how fast ILLIAC IV can perform these operations.

A simulator program (NISIM) for the Ballistic Missile Defense oriented to the Nike X program has been developed by Bell Telephone Labs. The NISIM package programmed for a GE 635 computer is divided into several different simulation programs which cover: (1) environment simulation, (2) hardware interface simulation, (3) function task simulation, (4) Nike X simulation control, and (5) report generation. The computer functions are oriented for a multi-processor configuration. The use of and the efficiency of ILLIAC IV in this type of radar environment is desired for BMD applications. The actual procedure describing how this evaluation is to be accomplished has not been completed.

REFERENCES

[1] El-Assar, Rateb. Numerical Solution of the Boundary Layer Equations with Slip Boundary Conditions: Part I. ILLIAC IV Document Number 189, (June 14, 1968).

[2] LaFrance, Jacques. Implementation of "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis" on ILLIAC IV. ILLIAC IV Document Number 166, (November 29, 1967).

[3] Nordsieck, A., and Hicks, B. L. Monte Carlo Evaluation of the Boltzmann Collision Integral. Coordinated Science Laboratory Report R-307, (1966).

[4] Benokraitis, Vitalius. Alternate Storage Methods for Two-Dimensional Hydrodynamics Calculations. ILLIAC IV Document Number 190, (May 27, 1968).

[5] Troyer, Stephen R. Sparse Matrix Multiplication. ILLIAC IV Document Number 191, (June 6, 1968).

[6] Sameh, Ahmed, and Han, Luke. Eigen-Value Problems. ILLIAC IV Document Number 182, (April 4, 1968).

[7] Knapp, M. A., Ackins, G. M., and Thomas, John. Application of ILLIAC IV to Urban Defense Radar Problem. ILLIAC IV Document Number 173, (February 21, 1968).

APPENDIX A
SECTION 1
GEOMETRY BLOCKS

GENERAL

SYNTAX.

```
<geometry block> := <geometry block head>;<geometry compound tail>
<geometry block head> := #GEOMETRY #BEGIN <geometry declaration> |
         <geometry block head>;<geometry declaration>
<geometry compound tail> := <geometry statement> #END |
         <geometry statement>;<geometry compound tail>
```

Examples and SEMANTICS

The general syntax and semantics of the geometry block should be clear; if there is any question, see Section 5 Program, Blocks, etc., of the Burroughs ALGOL Manual, since the structure is very similar to a subset of that syntax. It is not yet certain whether geometry specifications will be allowed to be dynamic. Therefore at present, "the" geometry block cannot be labeled and will probably appear after the declarations and before the first statement of only the outermost block.

> NOTE: As a general rule, all geometry syntax will have syntax and semantics analogous to that which appears in the Burroughs ALGOL Manual or the TRANQUIL Report if "geometry" or "geometric" is left out of the metalinguistic variable name; e.g., <geometry block> is very similar to <block>. So general statements regarding semantics in the Burroughs ALGOL Manual will hold for geometry specifications.

But regardless, a geometry block can be considered to be a declaration of certain variables (geometric figures, sets, arrays) and assignment of values to some of these (figures and sets). The output of the section consists of geometric arrays; the same as ordinary arrays in that they can be typed as real, integer, etc., but different in that the valid indices range over geometric sets rather than sets defined by bound pair lists, i.e., over non-rectangular arrays, and geometric sets which are special index sets defined, i.e., given values, in a special way, and which can be used for control just like ordinary sets. No values are assigned to the geometric array variables in the geometry block, however.

SECTION 2

GEOMETRY DECLARATIONS

GENERAL

SYNTAX.

```
<geometry declaration> := <geometric figure declaration> |
          <geometric set declaration> |
          <geometric array declaration>
```

SEMANTICS

The purpose of a declaration is to define the characteristics of a quantity and assign an identifier to the quantity so that it may be referenced. Geometry declarations do this with geometric quantities: geometric figures, sets, and arrays. They do not assign values to these quantities; that is done by geometry statements. These declarations will have scope within the block in which the geometry block appears in the same way that regular set and array declarations have scope within the block in which they occur.

GEOMETRIC FIGURE DECLARATIONS

SYNTAX.

```
<geometric figure declaration> := <geometric coordinate type>
          <geometric figure type> <geometric figure list> |
          <geometric figure type> <geometric figure list>
geometric figure type> := #CURVE | #CURVE (<dimension number>) |
          #SURFACE | #SURFACE (<dimension number>) |
          #REGION (<dimension number>)
<geometry list> := <geometry segment> |
          <geometry list> , <geometry segment>
<geometry segment> := <geometry identifier> |
          <geometry identifier> [<bound pair list>]
<geometry identifier> := <identifier>
<identifier> := {see Burroughs ALGOL Manual}
```

<bound pair list> := {see Burroughs ALGOL Manual}
<dimension number> := {see Tranquil Report}
<geometric coordinate type> := #CARTESIAN | #POLAR #SPHERICAL

<u>Examples</u>:

Geometric Figure Declarations:

```
#CARTESIAN #CURVE A, CURVE [1:2], SEMICIRCLE [1:2,1:10];
#SPHERICAL #SURFACE TOP [1:2], HEMISPHERE, BARNDOOR [1:100];
#REGION (2)  WALDENPOND, USA [-1:3];
```

SEMANTICS.

The #CURVE declaration says that A (not subscripted) and CURVE and SEMICIRCLE (both subscripted) are variables whose values, when assigned, will be curves (line segments or sections of parabolas, etc., or some arbitrary curve) in a two-dimensional space. The #SURFACE declaration does the same for surfaces in three-space. The region declaration says that WALDENPOND (there is only one WALDENPOND) is the name of some region, value to be assigned later, in two-space (a quantity which has area). Regions can be declared in three-space. The bound pair list signifies that there is an array of regions (or curves or surfaces) each of which may have a different value; e.g., USA [-1] may be a square with coordinates (0,0), (1,0), (1,1), (0,1), USA [0] may be a circle radius 1, center at the origin, USA [{3,4}] may be undefined. Geometric figures are mathematically continuous objects (although regions can be disconnected). Curves in two-space and surfaces in three-space will be used to build regions in two- and three-space, respectively. Geometric figures will eventually be defined with reference to a regular coordinate system, e.g., cartesian or polar in two-space, cartesian, spherical or possibly cylindrical in three-space. These coordinate systems are considered to extend through the whole space, i.e., they are infinite in extent.

GEOMETRIC SET DECLARATIONS

SYNTAX.

```
<geometric set declaration> := <geometric coordinate type> <geometric
          set type> <geometric set list> | <geometric set type>
          <geometric set list>
<geometric set type> := #GRID (<dimension number>) |
                        #MESH (<dimension number>) |
                        #BOUNDARY (<dimension number>) |
                        #INTERFACE (<dimension number>)
<geometric set list> := <geometric set segment> |
          <geometric set list> , <geometric set segment>
<geometric set segment> := <geometric set identifier> |
          <geometric set identifier> [<bound pair list>]
<geometric set identifier> :=
```

Examples:

Geometric Set Declarations:

```
#GRID(2) CARTESIAN, POLAR [1:2], G;
#SPHERICAL #MESH(3) ATMOSPHERE, GRID, M[1:10];
#CARTESIAN #BOUNDARY(2) POLAR, SEMICIRCLE [0:1];
#INTERFACE(3) HEMISPHERE [0:3], BOUNDARY;
```

SEMANTICS.

The #GRID declaration says that CARTESIAN, POLAR [{-1,0,1}], and G are variables whose values are grids in two-space (it does not refer to characteristics of these grids, e.g., CARTESIAN is not necessarily a cartesian grid).

The #MESH, etc., declarations do the same for meshes, boundaries, and interfaces. To define these entities: A grid is essentially a coordinate system defined with reference to a regular coordinate system (see GEOMETRIC FIGURE DECLARATIONS. Semantics.) which covers a finite position of the space. This grid may itself be a familiar type of coordinate system, i.e., cartesian, spherical, etc., possibly with new unit lengths, new unit angles, or it may be a cartesian-like system with axes skewed, rotated, and translated, or some other unusual system. A grid can perhaps best be thought of in simple cases as a set of lines at unit distances (or possibly unit angles) from each other with the set of "grid points" at the intersections being special points. A mesh then is the "intersection" of a region, defined with reference to a regular coordinate system, with a grid, defined with reference to a regular coordinate system, giving a set of grid points which lie "within" the region and the points where the grid lines intersect with the boundary (here the "real" mathematically continuous boundary) of the region. The mesh can be redefined to exclude or include other points; such as, the points where the boundary of the region intersects grid lines but not grid points, or points just outside the boundary, respectively. A boundary then is a set of points where a <u>section</u> of the "real" boundary of the region intersects the grid lines, or grid points within a certain distance, defined with reference to the grid coordinate system and in terms of the grid unit distances. The section of the "real" boundary could be the whole boundary if boundary conditions are the same for all points on the boundary. In the same way that a boundary is a special set of points connected with the "edge" of a region, interface is a special set of points connected with a curve or surface internal to the region where, as in the case of boundary points, special calculations need to take place. It is also possible that the subregions, into which the curve divides the region, may have different meshes associated with them. An interface will be used to relate them.

GEOMETRIC ARRAY DECLARATIONS

SYNTAX.

```
<geometric array declaration> := <geometric array declarator>
                                 <storage> <geometric array list>
<geometric array declarator> := <type> #GEOMETRIC #ARRAY
<type> := <see Tranquil report>
<storage> := <see Tranquil report>
<geometric array list> := <geometric array segment> | <geometric array
                          list>, <geometric array segment>
<geometric array segment> := <geometric array identifier> |
                             <geometric array identifier>
                             (<geometric set indicator>)
<geometric set indicator> := <geometric set identifier> | <geometric
                              set identifier> [<subscript list>] |
                             <geometric set identifier>, <geometric
                              set indicator> | <geometric set identifier>
                             [<subscript list>], <geometric set indicator>
<subscript list> := <see Tranquil report>
```

Examples.

```
        #REAL #GEOMETRIC #ARRAY
        WINDEW, WINDNS, WINDV, TEMP, PRESSURE,
        HUMIDITY (ATMOSPHERE), STRESS,
        STRAIN (JOINT [1]), FRAME;
```

SEMANTICS.

The geometric array declaration is like an ordinary array declaration in that it allows values to be assigned to an array name over a certain range of index values. It differs in that this range of index values cannot, in general, be described by a bound pair list; i.e., it is not usually "rectangular". The range of index values allowed in one dimension (of this multi-dimensional index set) is generally a non-constant function of the values in the other dimensions. These "non-rectangular" sets correspond to the grid points which lie inside a certain geometric region (or possibly just slightly outside). In the example WINDEW, ... HUMIDITY, all are array variables capable of having a value corresponding to each of a set of index values, that set being the geometric set ATMOSPHERE. STRESS and STRAIN are arrays defined over the set of index values equal to the union of the geometric sets in the list in parantheses; i.e., JOINT [1], and FRAME. It should be noted that if the array corresponding to one set in the list is to be independent of the array corresponding to another set in the list, then the intersection of those two geometric sets should be the null set.

JAN

JAN 29 1973

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA

3 0112 002022835